Large flexible software systems tend to incur “bloat”, here defined as the runtime overhead induced by the accumulation of excess functionality and objects. Removing bloat is hard as these overheads are a side-effect of the same trends that have fuelled software growth. Even defining and measuring bloat is non-trivial, as software doesn’t come with built-in labels that indicate which portions of computation are necessary for a given application and which lead to bloat. Much progress has been made in novel analysis tools that aid (human experts in) the process of finding bloat by highlighting signs of excessive activity and data flow. However, there has been very little research focus on understanding the connection between sources of bloat and its system level implications.

In particular, excess resource usage due to bloat could be a significant source of power-performance inefficiencies, but the relation between bloat and energy efficient design remains unexplored. In order to systematically devise effective mechanisms for reaping power-performance benefits through bloat mitigation, we require a deeper insight into exactly when excess features can originate bloat, when the resource overheads of bloat are most pronounced and when bloat matters for power performance. This dissertation explores the problem of software bloat and its energy efficiency implications from multiple perspectives to develop a better understanding of these connections.

First, we establish the need for a whole systems perspective in assessing potential energy efficiency benefits of bloat reduction, based on a systematic empirical and analytical study that highlights a curious interplay between bloat, energy proportionality and system bottlenecks. Second, we present a novel static analysis algorithm to perform an automated code transformation for object reuse that mitigates bloat involving the generation of excess temporary objects within loops. Third, we introduce the idea of concern augmented program analysis (CAPA), to identify sources of bloat due to excess features; the technique uses externally supplied information about program concerns and their properties as an abstraction of underlying intent. Fourth, as an early diagnostic aid, we use a statistical topic model to automatically discover latent concerns from source code statements and then correlate these latent concerns with resource usage properties that vary at statement granularity. The statistical model has a built-in sensitivity to the context of individual statements so that it can discover even diffused concerns without any apriori concern information.

Together, our findings show that presence of excess features, in itself, may not lead to (runtime) bloat, unless these features have some structural interaction with essential features. Further, the overheads due to such structural interactions, in turn, may not cause substantial bloat in the resource consumption of a long running (server) application unless incurred repeatedly during program execution. Finally, even such bloated resource usage has a pronounced impact on power-performance only if it affects a system bottleneck or a hardware resource that has a high degree of energy proportionality and consumes a high fraction of power compared to the other system resources.

We conclude that energy wastage due to bloat need not be an inevitable consequence of over-provisioning flexibility. Instead, the extent to which excess features result in runtime bloat and poor power-performance is determined by certain characteristics of the program structure and of the underlying hardware system -- these represent potential control points that could be exercised to develop principled design approaches for mitigating bloat without sacrificing flexibility or productivity.